XM-tree, un nuevo índice para Recuperación de Información en la Web

Claudia Deco, Guillermo Pierángeli, Cristina Bender
Departamento de Sistemas e Informática
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
(2000) Rosario, Argentina
{deco, bender}@fceia.unr.edu.ar

y

Nora Reyes
Departamento de Informática,
Universidad Nacional de San Luis
(5700), San Luis, Argentina
nreyes@unsl.edu.ar

Abstract

Web Information Retrieval is another problem of searching elements of a set which are closest to a given query under a certain similarity criterion. It is of interest to take advantage of metric spaces in order to solve a search in an effective and efficient way. In this article, we present an extension of the M-tree index, called *XM-tree*, in order to improve search results. This index allows dynamic insertion of new data, reduces search costs using prunings and precalculated distances, and uses a tolerable amount of space, which makes this index apt for the extensive and dynamic Web. The proposed extension indexes Web documents, uses L_2 as indexing distance and L_∞ as similarity criterion to solve queries. We also present experiments validating the results.

Keywords: Metric Spaces, Similarity Searching, M-tree, XM-Tree

Resumen

La Recuperación de Información de la Web es uno más de los problemas de buscar en un conjunto los elementos más cercanos a una consulta dada bajo un cierto criterio de similitud. Es de interés aprovechar las cualidades de los espacios métricos con el objeto de resolver una consulta de manera efectiva y eficiente. En este artículo, se presenta una propuesta de búsqueda utilizando *XM-tree* que es una extensión del índice M-tree. Este índice permite la inserción dinámica de nuevos datos, reduce los costos de búsqueda con distancias precalculadas y podas, y utiliza una cantidad de espacio tolerable, lo que lo hace apto para el extenso y dinámico entorno Web. La extensión propuesta indexa documentos Web, emplea L_2 como distancia de indexado y resuelve las búsquedas aplicando como criterio de similitud la norma L_∞. Se presenta la experimentación que valida los resultados.

Palabras claves: Espacios Métricos, Búsqueda por Similitud, M-tree, XM-Tree

1 INTRODUCCIÓN

La búsqueda es un problema fundamental en las ciencias de la computación, presente en casi cualquier aplicación. La operación de búsqueda ha sido tradicionalmente aplicada a datos estructurados. Con la evolución de las tecnologías de información y comunicación han surgido repositorios de información no estructurada. Este es el caso de la web, que está constituida por millones de páginas. Los nuevos tipos de datos (texto, imágenes, audio, vídeo) son difíciles de estructurar, tanto manual como computacionalmente, lo que restringe los tipos de consultas a realizar. Un entorno de este tipo requiere algoritmos y modelos de búsquedas más generales. Surge así el concepto de *búsqueda por similitud* o *búsqueda por proximidad*, que consiste en recuperar los elementos similares o cercanos al elemento consultado. La similitud se modela con una *función distancia* que satisface la desigualdad triangular y el conjunto de objetos se llama espacio métrico. En algunas aplicaciones el espacio métrico se torna un tipo particular llamado espacio vectorial.

Muchos trabajos apuntan a disminuir el costo de calcular las distancias entre los objetos, y reducir la E/S, construyendo un índice, es decir una estructura de datos para reducir el número de evaluaciones de distancia en el momento de consulta. En [3] se presenta un marco unificado que describe y analiza las soluciones existentes a este problema, y se distinguen dos principales técnicas de indexado: una basada en pivotes y otra basada en particiones de Voronoi. Analizadas las distintas técnicas, se propone realizar una extensión al *M-tree*, para indexar los documentos de la web y acelerar las búsquedas descartando mayor cantidad de objetos irrelevantes.

El resto del trabajo se organiza de la siguiente forma: en la Sección 2 se presentan conceptos básicos de Recuperación de Información y de Espacios Métricos. En la Sección 3 se presentan trabajos relacionados. En la Sección 4 se propone el *XM-Tree* como una extensión al *M-Tree*. La Sección 5 presenta los resultados de la experimentación. Finalmente, se presentan las conclusiones.

2 CONCEPTOS BÁSICOS

2.1 Recuperación de Información

La Recuperación de Información, se define de la siguiente manera: dado un conjunto de documentos y una consulta, determinar el subconjunto de documentos relevante a la consulta [1]. Uno de sus desafíos es determinar la relevancia de un documento con respecto a la consulta formulada por un usuario. El proceso de búsqueda es realizado por un motor de búsqueda, el cual maneja un cuerpo de documentos a través de dos pasos: indexado y recuperación. El indexado produce un índice invertido, que se construye a partir del cuerpo de documentos realizando un mapeo entre cada palabra presente en el cuerpo hacia los documentos que la contienen. La ocurrencia de un término en un documento se llama *posting*. El conjunto de postings asociados a un término se almacena en una *posting list*. En la recuperación, el motor de búsqueda recibe como entrada una consulta formada de una o más palabras. Un motor que utiliza índice invertido revisa qué documentos contienen las palabras ingresadas según el índice, y obtiene las posting lists de cada una. Estas listas se procesan para obtener el conjunto de resultados final, según la semántica de la consulta manejada por el motor. Por ejemplo, los motores booleanos exigen que la consulta se formule utilizando operadores booleanos entre las palabras ingresadas, y aplican las operaciones de conjunto para obtener el conjunto resultado.

El desempeño de un motor de búsqueda se puede evaluar a través de los indicadores *Precisión* y *Recall*. La Precisión es el número de documentos relevantes a la consulta dada dividido el total de documentos recuperados. El Recall es el cociente entre la cantidad de documentos recuperados y el total de documentos relevantes de la colección. Un motor alcanza un buen rendimiento cuando maximiza ambos valores; es decir, recupera la mayoría de los documentos relevantes disponibles en la colección con la menor cantidad de documentos irrelevantes.

Una forma de representar los documentos es el *modelo espacio vectorial*, en el cual los

documentos son representados con vectores de palabras. En estos vectores no se consideran términos muy frecuentes, artículos y pronombres. Además, los verbos conjugados, sustantivos y adjetivos, se reducen a una forma canónica con un proceso de *stemming*. Con el conjunto de palabras resultante se representa el documento como un vector en el espacio euclídeo. Cada término canónico representa un eje en este espacio. Si el término t_i ocurre en el documento d, la i-ésima componente del vector de d vale 1, si no vale 0. Las consultas se representan de la misma manera. La similitud entre una consulta q y un documento d se cuantifica calculando el producto interno entre los vectores de q y d: $sim(q,d) = \sum q_i \times d_i$. Esto permite retornar una secuencia de documentos ordenados por su relevancia a la consulta, ofreciendo un mecanismo de ranking de resultados.

Esta representación no captura la distinta importancia entre un término y otro en un mismo documento. La solución es que cada componente represente el peso del término en el documento. Este peso puede ser el número de ocurrencias, pero esta representación favorece a los documentos más grandes. Esto se resuelve empleando el *esquema TF,* o de frecuencia de términos, el cual consiste en normalizar los vectores dividiendo cada componente por la longitud de los mismos. Así, cada componente d_i del vector vale tf_i, o sea la frecuencia del término t_i en el documento d.

El esquema TF no tiene en cuenta la distribución del término en la colección entera. Esto se soluciona con la noción de frecuencia de documento inversa: $idf_i = log\ (N / n_i)$, donde N es el tamaño de la colección y n_i es el número de documentos donde ocurre el término i-ésimo. En la práctica, se combinan las medidas *tf* e *idf* en el *esquema TF/IDF*: $w_{ij} = tf_{ij}\ x\ idf_i$ donde w_{ij} es el valor que se le asigna a la componente i del documento j.

2.2 Espacios Métricos

Un espacio métrico (X, d) está formado por un universo de objetos válidos X y una función de distancia d: $X \times X \rightarrow R^+$ que mide la semejanza o proximidad entre dos objetos cualesquiera. La función distancia tiene las siguientes propiedades: no negatividad ($d(x, y) \geq 0$), simetría ($d(x, y) = d(y, x)$), reflexividad ($d(x, x) = 0$), positividad estricta ($\forall x, y \in X, x \neq y \Rightarrow d(x, y) > 0$), y satisface la desigualdad triangular ($d(x, z) \leq d(x, y) + d(y, z)$). El subconjunto finito U de X, de tamaño $n = |U|$, es el conjunto donde se realizarán las búsquedas, y es llamado el diccionario o base de datos. Los tipos de consultas de interés en los espacios métricos son la *Consulta por rango* $(q, r)_d$ que recupera todos los elementos que están dentro de la distancia r de q, *o* sea $\{ u \in U \mid d(q, u) \leq r \}$ y la *Consulta por k vecinos más cercanos* $NN_k(q)$ que recupera los k elementos más cercanos a q en U, obteniendo un conjunto $A \subseteq U$ tal que $|A| = k$ y $\forall u \in U, v \in U - A, d(q, u) \leq d(q, v)$. En este trabajo se utiliza la *Consulta por rango estricta* $(q, r)_d$ para recuperar todos los elementos que están dentro de la distancia r de q y no la alcanzan, o sea $\{u \in U \mid d(q, u) < r\}$.

Si los elementos del espacio métrico (X, d) son tuplas de números reales, el par es llamado *espacio vectorial*. Entonces, un espacio vectorial k-dimensional es un espacio métrico particular donde los objetos se identifican con k coordenadas de valores reales $(x_1, ..., x_k)$. En los espacios vectoriales, hay varias funciones de distancias definidas. Las más utilizadas son la familia de distancias L_s, definidas como: $L_s((x_1, ..., x_k), (y_1, ..., y_k)) = (\sum | x_i - y_i |^s)^{1/s}$. La distancia L_1, también llamada *de bloques*, es la suma de las diferencias entre las coordenadas. La distancia L_2, o *distancia euclídea*, corresponde a la noción de distancia espacial. Otra muy utilizada es L_∞, que equivale al límite de L_s cuando s tiende a infinito, y se define como la máxima diferencia entre dos coordenadas de los dos objetos dados: $L_\infty((x_1, ..., x_k), (y_1, ..., y_k)) = \text{máx} \mid x_i - y_i \mid$.

Los índices sobre espacios métricos permiten la recuperación efectiva y eficiente de objetos. Efectiva, porque los resultados tienen un alto grado de exactitud por propiedades del espacio y del índice; y eficiente porque los índices se construyen para reducir el número de cálculos y objetos revisados. Estas dos propiedades son muy deseadas en un motor de búsqueda y hacen de la utilización de espacios métricos una alternativa interesante en los procesos de indexado y búsqueda.

En [3] se presenta un modelo unificado de los algoritmos de indexado de espacios métricos.

Todos los algoritmos de indexado particionan el conjunto U en subconjuntos, y construyen un índice para determinar los subconjuntos candidatos donde los elementos relevantes a la consulta podrían aparecer. Los algoritmos de indexado se dividen en dos clases principales: basados *en pivotes* y de tipo Voronoi (o basados en particiones compactas). Los primeros trabajan tomando k puntos llamados *pivotes* y mapean el espacio métrico sobre R^k usando la distancia L_∞. Los algoritmos tipo Voronoi particionan el espacio métrico en subconjuntos según su cercanía a ciertos puntos llamados *centros*. Los algoritmos basados en pivotes necesitan mucho más espacio para almacenar las clases resultantes de los k pivotes que el que utilizan los de tipo Voronoi para el mismo número de particiones. Cuando los algoritmos basados en pivotes cuentan con la memoria necesaria y usan el número óptimo de pivotes, la cota inferior del costo de búsqueda es mejor que la cota de los de tipo Voronoi. Esta propiedad se desvanece a medida que la dimensión intrínseca del espacio métrico crece, es decir, cuando es más difícil buscar en dicho espacio.

La web es un repositorio de tamaño continuamente creciente. Como en este trabajo se pretende discriminar las páginas por su contenido, se plantea representarlas con vectores de sus palabras representativas, lo que presupone el manejo de vectores con miles de componentes. Esto no permite asegurar que la dimensión intrínseca del espacio métrico será alta o no, pero sí que soluciones para espacios vectoriales no son aplicables. Considerando la incertidumbre sobre la dimensión intrínseca del espacio, los algoritmos de tipo Voronoi aparecen como más aptos. Además, el ambiente de la web es muy dinámico, continuamente se están agregando, modificando y borrando páginas. Por esto, el índice propuesto permite la inserción de datos. En este trabajo se optó por extender el algoritmo de indexado M-tree, dado que éste mejora algoritmos anteriores, permite inserciones y el marcado en las hojas de un elemento como eliminado.

2.3 M-tree

El método de acceso *M-tree* [4] es propuesto para organizar y buscar en grandes conjuntos de datos. Los experimentos muestran un comportamiento razonablemente bueno en espacios de alta dimensión y conjuntos de datos de tamaño creciente. El M-tree es un árbol paginado, dinámico y balanceado, que permite insertar y borrar datos de manera dinámica*, manteniéndose balanceado sin la necesidad de tener que hacer reorganizaciones periódicas. Los nodos del M-tree son de tamaño fijo y permiten un número máximo M de nodos hijos. Los nodos hoja almacenan todos los objetos indexados de la base de datos y los nodos internos, llamados *objetos ruteadores*, mantienen objetos de la base de datos con un rol de ruteo asignado por el algoritmo de promoción específico.

Cada nodo N corresponde a una región del espacio métrico indexado (X, d), definida como $Reg(N) = \{ O \in X \mid d(O_r, O) \leq r(O_r) \}$, donde O_r es el objeto ruteador del nodo N y $r(O_r)$ es el llamado *radio de cobertura*. Por definición, todos los objetos en el subárbol con raíz en N están dentro de la región $Reg(N)$. Si N es un *nodo interno*, mantiene como entradas el objeto ruteador O_r y el radio $r(O_r)$, también tiene un puntero asociado, $ptr(T(O_r))$, que referencia la raíz del subárbol, denotado $T(O_r)$ y llamado *árbol de cobertura*. Finalmente contiene una distancia a $P(O_r)$, su objeto padre, es decir el objeto ruteador que tiene a O_r como una de sus entradas. Esta distancia no está definida en la raíz del M-tree. Los *nodos hoja* son similares a los objetos ruteadores, pero el radio de cobertura no es necesario y un puntero almacena el identificador del objeto en la base de datos.

En [4] se presentan dos algoritmos de búsqueda por similitud: consulta por rango y consulta de k vecinos más cercanos. El objetivo de ambos es reducir la cantidad de nodos accedidos y la cantidad de distancias calculadas, utilizando las distancias precalculadas almacenadas en los nodos del M-tree, constituida por $d(O_r,P(O_r))$ y $r(O_r)$, aplicando la desigualdad triangular. Si bien tratan las consultas por rango como no estrictas, aquí se utilizan consultas por rango estrictas, que son un caso particular de las anteriores y sólo requieren pequeñas modificaciones en su tratamiento. La consulta

* La eliminación se realiza en las hojas, marcando el elemento como eliminado, por si se usa como objeto ruteador.

estricta $(Q, r(Q))_d$ selecciona todos los objetos de la base de datos tal que $d(O_j, Q) < r(Q)$.

Para determinar los nodos relevantes a la consulta utiliza el algoritmo *RangeSearch*, que parte del nodo raíz y recorre recursivamente todos los caminos que no pueden ser excluidos hasta los nodos hoja con objetos que satisfacen la consulta. Cuando el algoritmo accede a los objetos O_r del nodo N, la distancia del objeto padre O_p a la consulta se calcula una sola vez (las distancias de O_r a O_p están precalculadas en el índice) así es posible podar subárboles sin calcular más distancias.

3 TRABAJOS RELACIONADOS

El índice sobre espacios métricos M-tree fue mejorado en varias versiones posteriores con el fin de obtener mayor velocidad de indexado y recuperación. El *Slim-tree* [8] es una versión que acelera el indexado con el método de separación *minimal spanning tree*, y reduce la superposición de regiones con el algoritmo *Slim-down*, mejorando la performance de las consultas. El *M^+-tree* [10] introduce el concepto de *dimensión clave*: particiona el espacio ocupado por un árbol en dos subespacios no superpuestos llamados *nodos gemelos*. En la búsqueda, la dimensión escogida como clave evita calcular distancias entre los objetos y por lo tanto, aumenta la efectividad de las podas. El *BM^+-tree* [11] extiende el M^+-tree mejorando el filtrado. En lugar de una dimensión clave utiliza dos para construir un *hiperplano binario* que particiona mejor los datos. El *Density-Based Metric tree* [9], o *DBM-tree*, minimiza la superposición en nodos de alta densidad, relajando el requerimiento de altura balanceada del árbol. La altura es mayor en las regiones más densas. Logra mayor desempeño en las búsquedas porque el árbol está ajustado a la distribución de los datos en las distintas regiones. El *DBM*-tree* [7] extiende al anterior, agregando en cada nodo una matriz con algunas distancias precalculadas entre los objetos que contiene con el fin de aumentar la velocidad de indexado y de búsqueda, porque permiten mayor número de podas de datos irrelevantes. En [5] se introduce los principios básicos y los resultados de una experimentación del *M^2-tree*, un índice paginado y balanceado, que se construye sobre varios espacios métricos (atributos) a la vez y en una única estructura, permitiendo resolver eficientemente consultas que combinan arbitrariamente dichos atributos. El *XM-tree* propuesto aquí es un caso muy particular de este último índice, porque extiende un M-tree sobre espacios vectoriales, adaptando sus algoritmos a la búsqueda en la Web.

Con respecto a los motores de búsqueda en la web, su continuo crecimiento propone un desafío constante a la creación de índices veloces. Una solución actual es la utilización de índices invertidos, empleados por buscadores reconocidos como Google [2]. Sin embargo, estos índices presentan una desventaja, consecuencia de su estructura intrínseca: indexan de a una palabra. Cuando se resuelven consultas de varias palabras, primero se recuperan los documentos correspondientes a cada una de éstas, y luego éstos se someten a un proceso de unión o intersección. Para agilizar la velocidad de respuesta estos buscadores devuelven no sólo los documentos que satisfacen la estrategia de búsqueda, sino que devuelve también aquellos que posean alguno de los términos. El indexado con el XM-Tree propuesto en este trabajo mejora estos resultados.

4 PROPUESTA

El objetivo de este trabajo es utilizar las propiedades de los índices sobre espacios métricos para mejorar la calidad de los resultados de una búsqueda de información. El análisis sobre algoritmos de indexado sobre espacios métricos permitió vislumbrar la aptitud del M-tree para el entorno Web. Se optó entonces por extender el M-tree ya que al realizar una búsqueda, elige los recorridos analizando la consulta ingresada con información que contiene su estructura. Además, logra velocidad porque en este proceso va descartando los subárboles que no tienen datos próximos a la consulta. Los resultados obtenidos son correctos porque en el proceso de poda los subárboles rechazados contienen siempre datos irrelevantes, por definición del M-tree. Para alcanzar ambos

objetivos, velocidad y calidad de resultados, se elige una adecuada representación de las páginas Web y un conveniente criterio de similitud entre las mismas que posibiliten el empleo de un M-tree. A continuación se presentan las decisiones tomadas para esta propuesta.

Generación del Diccionario: El diccionario es un archivo de texto para medir las ocurrencias de las palabras en los documentos. Cada entrada corresponde a un término. Llamaremos *T_DIC* a la dimensión del vector diccionario.

Representación de los Documentos: Los documentos se representan como puntos en un espacio vectorial, cuya dimensión es el tamaño T_DIC del diccionario. Cada eje de este espacio representa una entrada del diccionario. Se utiliza el esquema TF para representar el documento[†]. El esquema TF es eficaz porque los documentos que comparten palabras muy frecuentes y del diccionario, quedan representados por vectores muy cercanos en cualquier distancia L_s. En dicha situación, un M-tree que cuenta como métrica una distancia L_s, agrupa estos documentos en el mismo subárbol y así acelera la resolución de las consultas conformadas por los términos compartidos.

Indexado: El índice utilizado es un M-tree cuya distancia de indexado es L_2, lo que le permite agrupar satisfactoriamente los documentos similares en el esquema TF. Se prefirió L_2 a las normas L_∞ y L_1 porque es más exacta en un espacio vectorial.

Búsquedas: Las consultas están conformadas por una o varias palabras, y se representan con un vector q de dimensión T_DIC con valores reales. Los términos que figuran en la entrada t del diccionario se indican con un valor no nulo en la componente t-ésima de q, el resto de las componentes se evalúan en 0. El principal objetivo del sistema es recuperar los documentos donde co-ocurran todas las palabras (o sus variantes morfológicas) ingresadas en la consulta. El algoritmo de búsqueda del M-tree obtiene los documentos cuyos vectores de frecuencias sean más próximos a q, según la norma L_2, por lo que hay que escoger valores adecuados para las componentes no nulas de q. Esto conlleva a calcular un q lo suficientemente cercano en el espacio euclídeo a todos los documentos que contengan los términos de la consulta y, de esta manera, permita recuperarlos en su totalidad. Como las frecuencias de mismas entradas pueden ser muy desproporcionadas entre sí en los documentos que las contienen, resulta imposible la obtención de un q óptimo porque habría que conocer de antemano los documentos que se pretenden recuperar. La situación se agrava si se pretende hacer una consulta por rango: ¿qué rango $r(Q)$ calcular? Ante estas circunstancias, se decidió utilizar un criterio de similitud entre los documentos y la consulta más relajado que L_2.

Para obtener los documentos que contengan todos los términos de la consulta, basta con chequear que las componentes que corresponden a dichos términos sean no nulas, las restantes componentes se pueden obviar. La resta entre 1 y la componente plasma este análisis: si el resultado es 1, ni la palabra ni sus variantes morfológicas figuran en el documento; si la diferencia es menor que 1 entonces el documento sí contiene al término. Entonces, el criterio de similitud es: asignar 1 a los elementos de q que corresponden a términos ingresados según el diccionario, calcular las restas de cada uno de dichos elementos y el correspondiente al vector del documento; si todas las restas dan menos que 1 entonces el documento satisface la consulta. Para extender este modelo al resto de las componentes fuera de la consulta y sin alterar los resultados, simplemente se les resta 0, o sea la componente de q. Como las frecuencias por definición varían entre 0 y 1 y entre todas suman 1, estas diferencias no alcanzarían al valor 1 (salvo un caso extremo) y no afectarían el criterio.

El nuevo criterio de similitud entre documentos y consultas se puede formular con el cálculo de la norma L_∞ entre el vector del documento y el vector q de la consulta. Si el resultado es menor que 1 entonces el documento contiene todos los términos de la consulta, en caso contrario, se descarta. Como el algoritmo de búsqueda del M-tree por definición utiliza la misma distancia con que se construyó el índice, para usar el criterio propuesto habría que armar el árbol empleando la norma L_∞

† Se descartó usar el esquema TF/IDF porque requiere para el cálculo de cada componente el número de ocurrencias del término en el resto de los documentos. Así, al ingresar un nuevo documento, habría que recalcular los vectores de todos los documentos, por lo que el índice no sería dinámico.

como métrica. Esta opción no resulta atractiva porque determina la proximidad de dos documentos teniendo en cuenta la frecuencia de un solo término, aquél cuya diferencia de frecuencias en ambos documentos es máxima, pudiendo llegar a considerar cercanos documentos que no comparten términos. La otra alternativa es seguir empleando como distancia de indexado a la norma L_2 pero modificar el algoritmo de búsqueda para que compare los documentos y la consulta con L_∞. Este cambio se traslada al criterio de poda que no podrá seguir usando L_2, si no que requerirá cierta información adicional para dirimir con una distancia distinta a la de indexado. Dicha información adicional concierne a las *componentes* de los datos del subárbol tratadas por separado, por lo que debe estar en el nodo interno raíz del subárbol analizado. Los cambios mencionados en el algoritmo de búsqueda y en la estructura de los nodos constituyen la propuesta de extensión del M-tree.

4.1 Propuesta de M-tree Extendido: XM-tree

El *XM-tree* es un índice sobre espacios vectoriales para la búsqueda en la web. Es una extensión del M-tree con la particularidad que trata las componentes de los vectores indexados por separado. Las extensiones sobre el M-tree original comprenden información adicional en los nodos y las correspondientes modificaciones en los algoritmos de construcción para mantenerla. La información que se añade hace referencia a las distancias entre las componentes de los datos indexados tratadas separadamente, con el fin de preservar estas medidas y emplearlas para posibles podas en el algoritmo de búsqueda. El objetivo de esta propuesta es lograr un índice sobre páginas web que utilice la norma L_2 como distancia de indexado y la norma L_∞ como distancia de consulta.

Se agrega al nodo interno el *vector de radios de cobertura* $rv(O_r)$. Si n es la dimensión de los datos, O_r y O_j tienen componentes O_{ri} y O_{ji} respectivamente, y consideramos una distancia f sobre el espacio de las componentes, debe valer para cada componente que $rv_i(O_r)$: $f(O_{ri}, O_{ji}) \leq rv_i(O_r)$ $i = 1, 2, ..., n$ $\forall O_j \in T(O_r)$. La distancia entre componentes f es una norma L_s, que, como se trata de una sola dimensión, para cualquier s coincide con el valor absoluto de la diferencia. Además se agrega el *vector de distancias* $dv(O_r, P(O_r))$, con las distancias entre las componentes del objeto ruteador O_r y las del padre $P(O_r)$. Cada componente vale: $dv_i(O_r,P(O_r)) = f(O_{ri},P(O_r)_i)$, $i = 1,2, ..., n$.

Entonces *cada nodo interno* del XM-tree queda conformado de la siguiente manera:

O_r	objeto ruteador
$ptr(T(O_r))$	puntero a la raíz de $T(O_r)$
$r(O_r)$	radio de cobertura de O_r
$rv(O_r)$	*vector con radios de cobertura de las componentes de O_r*
$d(O_r, P(O_r))$	distancia de O_r a su padre
$dv(O_r, P(O_r))$	*vector con distancias entre componentes de O_r a su padre*

Los nodos hoja no sufren modificaciones respecto al M-tree. Las entradas son: O_j*:* objeto de la base de datos, $oid(O_j)$: identificador del objeto y $d(O_j, P(O_j))$: distancia de O_j a su padre. Al igual que el M-tree, el objetivo de los algoritmos de búsqueda de esta extensión es reducir el número de nodos accedidos y de distancias calculadas, gracias a las distancias precalculadas almacenadas en $rv(O_r)$ y $dv(O_r,P(O_r))$ de los nodos internos y a la desigualdad triangular.

El XM-tree opera sobre datos de un espacio vectorial de dimensión n, con una distancia f sobre el espacio de las componentes. Dados $Q = (Q_1, Q_2, ..., Q_n)$ a buscar en el espacio vectorial y un vector $rv(Q)$ de radios de búsqueda con componentes $rv_i(Q)$, la consulta estricta $(Q, rv(Q))_f$ selecciona todos los objetos O_j de la base de datos tal que $f(O_{ji}, Q_i) < rv_i(Q)$, $\forall i = 1, 2, ..., n$.

El algoritmo modificado *RangeSearch* parte del nodo raíz y recorre todos los subárboles no excluidos por el criterio de poda hasta los nodos hoja con objetos que satisfacen la consulta.

En la línea 3 el algoritmo accede a los objetos O_r del nodo N. Las distancias entre las componentes Q_i y las del objeto padre O_{pi} se calculan una sola vez para i de 1 a n, y las distancias entre O_{ri} a O_{pi} figuran precalculadas en el vector $dv(O_r, O_p)$. Así, se pueden podar subárboles sin

calcular más distancias. La poda se efectúa en la línea 6 si se da la condición: si $f(O_{ri}, Q_i) \geq rv_i(Q) + rv_i(O_r)$ entonces, para cada O_j en $T(O_r)$ vale $f(O_{ji}, Q) \geq rv_i(Q)$. De esta manera $T(O_r)$ puede ser descartado. En la línea 9 se chequean los nodos hoja con objetos O_j. Aquí también las distancias entre las componentes O_{pi} y Q_i se calculan una vez y los elementos $dv_i(O_j, O_p)$ figuran en la estructura. La búsqueda en subárboles irrelevantes se evita en las líneas 4 y 10 empleando el siguiente resultado, considerando que los radios de cobertura $rv_i(O_j)$ de los datos es 0: Si $|f(O_{pi}, Q_{ii}) - dv_i(O_r, O_p)| \geq rv_i(Q) + rv_i(O_r)$ entonces $f(O_{ri}, Q_{ii}) \geq rv_i(Q) + rv_i(O_r)$. Estas dos condiciones son válidas porque son casos particulares de lemas propuestos por [4].

Algoritmo RangeSearch(node N, query_object Q, search_radius rv(Q))

```
1.  Let Op be the parent object of node N
2.  If N is not a leaf then
3.    For each object Or in N do
4.      If (|f(Opi, Qi) – dvi(Or, Op)| < rvi(Q) + rvi(Or) i=1..n) then
5.        Compute f(Ori, Qi) i=1..n
6.        If (f(Ori, Qi) < rvi(Q) + rvi(Or) i=1..n) then
7.          RangeSearch(*ptr(T(Or)), Q, rv(Q))
8.  else   // N is a leaf
9.    For each node child Oj in N do
10.     If (|f(Opi, Qi) – dvi(Oj, Op)| < rvi(Q) i=1..n) then
11.       Compute f(Oji, Qi) i=1..n
12.       If (f(Oj, Qi) < rvi(Q) i=1..n) then add oid(Oj) to the result set
13. End
```

Los algoritmos de construcción del XM-tree, *Insert* y *Split*, son los del M-tree original, con el agregado de sentencias para el mantenimiento de los nuevos datos $rv(O_r)$ y $dv(O_r, P(O_r))$.

Para llevar a cabo las búsquedas, una vez representadas las páginas web con vectores del esquema TF, se construye el XM-tree empleando como distancia de indexado d la norma L_2 y como distancia entre componentes f una norma L_s, que, como se trata de una sola dimensión, para cualquier s coincide con el valor absoluto de la diferencia. El algoritmo de búsqueda se invoca pasando un vector de radios con componentes $rv_i(Q)$ de valor 1. Con esta táctica se logra el objetivo de indexar los datos con L_2 y buscar con L_∞, esto último a partir de la siguiente equivalencia entre el criterio de búsqueda propuesto y la consulta por rango estricta $(Q, rv(Q))_f$ del XM-tree: $L_\infty(Q, O) < 1 \leftrightarrow \max\{|Q_i - O_i|\} < 1 \leftrightarrow |Q_i - O_i| < 1 \leftrightarrow f(Q_i, O_i) < 1 \leftrightarrow f(Q_i, O_i) < rv_i(Q) \quad \forall i=1, 2, ..., n$ ‡.

5 EVALUACIÓN EXPERIMENTAL

5.1 Implementación

El sistema propuesto en este trabajo ha sido implementado en C^{++} utilizando DJGPP [6]. Se construyó un diccionario de 1000 entradas, y considerando las variantes morfológicas de cada entrada, el diccionario agrupa unos 7000 términos. Se calculan los vectores TF de las páginas web a indexar, considerando solamente el texto visible de las páginas. Se genera un archivo de salida con los siguientes datos para cada página: número identificador, vector de frecuencias, título, URL y ubicación del respaldo. Este prototipo cuenta con procedimientos que modelan los algoritmos de inserción, separación y búsqueda del XM-tree. Al iniciarse lee el archivo de datos generado, construye el árbol dinámicamente y solicita el ingreso de consultas vía línea de comandos. La aridad M del árbol es un valor fijo del programa. La implementación ofrece al usuario la posibilidad de utilizar operadores lógicos. El operador por defecto es 'AND'.

En la implementación del algoritmo de separación se escogieron las políticas m_RAD e

‡ El logro de esta equivalencia justifica la decisión de emplear la consulta por rango estricta, ya que la consulta por rango original no sería útil porque permitiría alcanzar el valor de los radios $r_{vi}(Q)$.

Hiperplano Generalizado para modelar los algoritmos de promoción y partición respectivamente. En m_RAD, el algoritmo "mínima suma de radios" promueve los objetos cuya suma de radios de cobertura $r(Op_1) + r(Op_2)$ es mínima. Hiperplano Generalizado asigna cada objeto $O_j \in S$ al objeto promocionado más próximo. Esta estrategia es no balanceada. Por el momento se han considerado sólo estas dos políticas, pero como trabajo futuro se podrían evaluar las otras políticas.

Una vez procesada la consulta, el prototipo presenta datos de los tres tipos de búsquedas: la búsqueda en el XM-tree; una búsqueda exhaustiva, con la consulta original sobre todas las páginas indexadas; y una búsqueda que emula un sistema de índices invertidos. En la interfaz se presenta información sobre la estructura del XM-tree construido: cantidad de nodos internos, de nodos hoja y de datos indexados. Para analizar la eficiencia de la búsqueda en el árbol se muestran: cantidad y porcentaje de nodos recorridos y de nodos podados. Para verificar la efectividad de la búsqueda en el XM-tree se muestran la cantidad de resultados, la Precisión y el Recall. Luego del proceso de filtrado, se muestran nuevamente cantidad de resultados, Precisión y Recall, los cuales serían los valores finales alcanzados por el sistema. La estrategia de búsqueda con la que se invocó al algoritmo, se compara con los vectores de todas las páginas usando el criterio de similitud L_∞. Aquellas páginas que verifican el criterio y no se retornaron en la búsqueda, o que no lo verifican y sí fueron retornadas, son falencias del XM-tree y su cantidad se muestra con el título 'errores'.

La búsqueda exhaustiva corresponde a la aplicación de la consulta ingresada al texto de cada página. El objetivo es mostrar la efectividad de las siguientes propuestas hechas en este trabajo: el esquema TF como mecanismo de representación de páginas, el criterio de similitud L_∞ entre página y consulta, y el XM-tree como método de indexado. La cantidad de resultados diferentes entre esta búsqueda y la principal se muestran en pantalla. El número de resultados aquí obtenidos se considera el número total de documentos relevantes en la colección y se emplea para calcular los indicadores Precisión y Recall. La última búsqueda encuentra los datos que recuperaría un sistema con índices invertidos. La diferencia entre los datos recuperados por la primera instancia de esta búsqueda y la búsqueda principal también se muestra en la interfaz. La intención de esta comparación es verificar la mayor eficiencia del XM-tree sobre los índices invertidos. Los datos se ordenan de acuerdo al máximo producto interno con los vectores q, de mayor a menor.

5.2 Experimentación

Para la experimentación se realizaron consultas en el prototipo, con el fin de mostrar la performance del sistema en un ambiente similar al entorno web real. Las páginas que conforman el cuerpo de documentos que indexa el XM-tree en este testing constituyen un conjunto muy pequeño en comparación con el total que compone la web indexada, pues su objetivo es solamente experimental. Por esta misma razón, se trató de construirlo de la manera más representativa posible.

Para simular de la mejor manera alcanzable la heterogeneidad temática que caracteriza a Internet en una muestra reducida, se recurrió a la recolección de páginas web reales que traten varios temas distintos pero que compartan algunos términos significativos. Los documentos que componen la muestra se obtuvieron mediante consultas formuladas a Google. El cuerpo de documentos construido contiene poco más de 200 páginas, es suficientemente representativo porque se obtuvieron de la web real, la gran mayoría concierne a 8 tópicos de bastante interés, y a su vez, algunos de estos temas pueden estar relacionados entre sí por compartir términos significativos.

Un usuario que pretende recuperar información sobre el cáncer de pulmón podría hacerlo a través de la consulta: (cáncer OR tumor) AND (pulmón OR pulmones OR pulmonar). Una vez ejecutadas las búsquedas la información mostrada es la siguiente:

```
=== BUSQUEDA EN EL SISTEMA BASADO EN XM-TREE ============================
estructura        : 8 ints, 47 hojas, 224 datos, total 279
revisados         : 7 ints (87.50 %) 14 hojas (29.79 %) 96 datos (42.86 %)
podados           : 30                errores   : 0
Resultados XM-tree : 29               Precision : 1.00          Recall : 1.00
```

```
=== COMPARACION CON BUSQUEDA CON INDICE INVERTIDO (173 resultados) ==========
no recuperados por el XM-tree          : 144
no recuperados por el indice invertido  : 0
=== COMPARACION CON BUSQUEDA EXHAUSTIVA (29 resultados) ====================
no recuperados por el XM-tree          : 0
no recuperados por la busqueda exhaustiva : 0
```

En el campo *estructura* se informa la cantidad de nodos internos, nodos hoja y datos del árbol construido, la cantidad de datos coincide con el total de documentos indexados, en este caso 224. En la siguiente línea se muestran la cantidad y porcentaje de nodos *revisados* de cada tipo en la búsqueda XM-tree. La revisión de un nodo implica el cálculo de normas *f* del XM-tree. Un bajo porcentaje de nodos revisados indica un buen agrupamiento de los datos similares en las hojas del XM-tree, por lo que el algoritmo de búsqueda encamina la consulta hacia pocas hojas y poda muchos nodos. La cantidad de nodos *podados* corresponde al número de subárboles descartados por el algoritmo RangeSearch siguiendo los criterios de poda, o sea, nodos internos y hojas sin datos relevantes. En este caso es bastante alto, 30 podas sobre 55 nodos (internos y hojas), por la alta especificidad de la consulta en la colección. El número *errores* es nulo porque la búsqueda en el árbol recuperó correctamente todos los datos que verifican el criterio de similitud L_∞ con la consulta. Los indicadores *Precisión* y *Recall* de la recuperación XM-tree en sus valores óptimos reflejan la alta efectividad de la búsqueda en dicho índice: obtuvo todos y sólo los documentos relevantes, como se verifica en la comparación con la búsqueda exhaustiva.

La búsqueda que emula un sistema de índices invertidos obtiene 173 datos correspondientes a las posting lists de los términos ingresados, los cuales posteriormente se someterían al proceso de mezclado e intersección. Este valor se compara con el número de resultados XM-tree, en este caso 29, que como es mucho menor verifica el mejor desempeño del XM-tree para la consulta ingresada.

El archivo de páginas se reordenó aleatoriamente, para analizar la dependencia o no de la perfomance del XM-tree respecto al orden de inserción de los datos. Para esto, se construyeron cinco conjuntos de datos con distintas ordenaciones de las páginas que lo conforman. Se obtuvieron los mismos valores de Precisión y Recall, para cada uno de los conjuntos de datos, por lo que no es importante el orden de los documentos de entrada. Posteriormente se procedió a analizar la eficiencia de la búsqueda en el árbol construido sobre cada conjunto.

En la Figura 1 cada línea corresponde a una de las consultas e indica el porcentaje de datos revisados en cada conjunto. Los conjuntos de datos son numerados de 1 a 5. Como se aprecia, una consulta no registra diferencias abruptas de porcentaje entre un conjunto y otro. Esto demuestra, acerca del grado de eficacia del indexado, que el árbol agrupa de la igual manera páginas similares sin importar el orden del conjunto sobre el que se construyó. Por lo tanto, los valores similares de Precisión, Recall y datos revisados permiten concluir que la eficiencia y eficacia del sistema es independiente del orden en que se indexan las páginas. El segundo aspecto analizado es la aridad *M* del árbol, o sea, el tamaño de cada nodo. En la Figura 2, se muestran los porcentajes de datos revisados en cada consulta para diferentes aridades consideradas: 5, 10, 15, 20 y 25.

En la Figura 3 se presentan las cantidades totales de nodos de distintos árboles con los valores mencionados de *M*. Se observa cómo a mayor *M*, el árbol ocupa menos espacio pero revisa más datos, porque las hojas contienen muchos datos y el algoritmo de búsqueda las chequea por completo. Una pérdida de eficiencia aún más importante es la menor capacidad de agrupar datos de un XM-tree con alta aridad, porque el agrupamiento de datos similares en subárboles distintos sucede luego de que un nodo se desborda durante la inserción de uno nuevo, lo que no se da a menudo en un árbol con nodos de gran tamaño. Así se concluye que, si bien la elección de *M* no afecta la calidad de los resultados se debe escoger, en principio, un valor intermedio de acuerdo al tamaño de la colección (aquél cuyo árbol ocupe una cantidad de espacio razonable). Con la sucesiva incorporación de nuevos datos a la colección y su inserción en el árbol, el aprovechamiento razonable de espacio mermará, pero no se perderá eficiencia en la resolución de consultas.

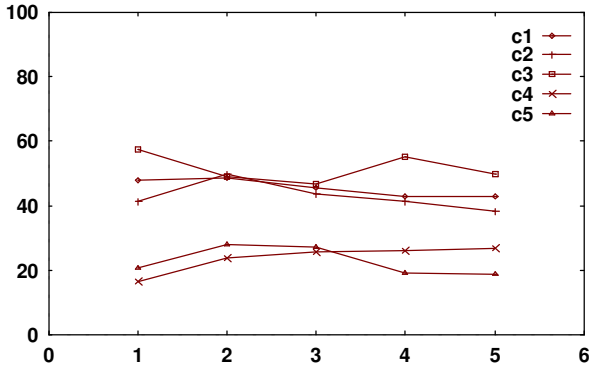

Figura 1. Porcentaje de datos revisados por cada conjunto de datos.

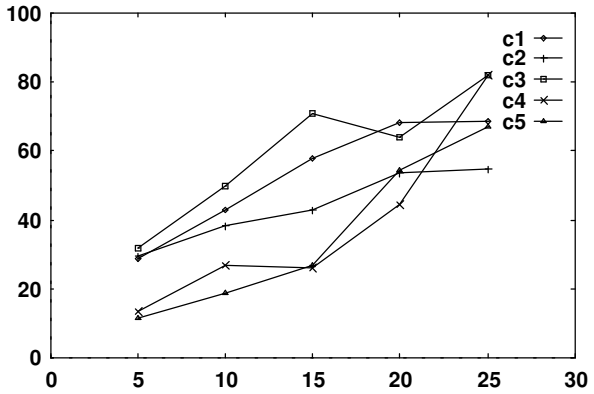

Figura 2. Porcentajes datos revisados por *M*.

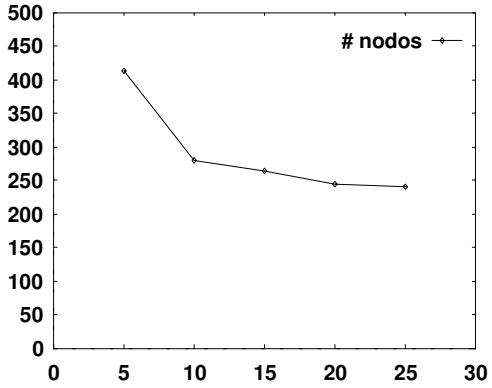

Figura 3. Número de nodos por aridad *M*

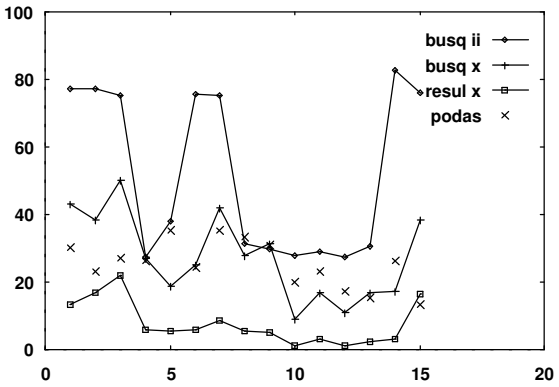

Figura 4. Porcentajes de datos revisados, resultados y podas por consulta

Para la siguiente etapa experimental se preservó el XM-tree de aridad 10, debido a que con un valor menor el desaprovechamiento de espacio ya es muy ostensible, como muestra la Figura 3 en $M = 5$. Con valores más altos, la revisión de datos no relevantes tiende a aumentar (Figura 2).

La última fase de la experimentación consistió en comparar la performance de este sistema y un sistema de índices invertidos. A las cinco consultas utilizadas para las primeras experiencias, se agregaron diez consultas que atañen al resto de los principales temas de la colección, incluso algunas muy específicas dentro de un mismo tema. En la Figura 4 las consultas están numeradas de 1 a 15 en las abscisas, para cada una se muestran los siguientes valores en las ordenadas: el porcentaje de datos recuperados por un sistema de índices invertidos; el porcentaje de datos revisados en la búsqueda del XM-tree; el porcentaje de datos recuperados del XM-tree; y el número de nodos podados por la búsqueda en el árbol. En dicha gráfica se observa cómo la cantidad de datos revisados por la búsqueda XM-tree es proporcional a la cantidad de resultados relevantes en cada consulta. Esto se debe al agrupamiento de documentos similares en las hojas del XM-tree, sólo se revisan las hojas con documentos próximos a la consulta. Esto no ocurre en los índices invertidos, donde la cantidad de datos revisados depende directamente del término ingresado con más ocurrencias en la colección, y que, por tal motivo, contiene posting lists de gran tamaño. En cuanto a la revisión de datos de la colección durante una búsqueda, el sistema propuesto realiza un gasto proporcional a la cantidad de documentos relevantes a la consulta.

6 CONCLUSIONES

Velocidad en la recuperación y calidad de los resultados son dos propiedades necesarias en cualquier sistema de Recuperación de Información. Los espacios métricos cuentan con índices que permiten la recuperación de objetos cercanos a uno dado de una forma rápida y bastante exacta, por

lo que resultan estructuras prometedoras sobre las cuales se pueden construir motores de búsqueda.

En este trabajo se propuso el XM-tree que es un índice sobre espacios vectoriales. Es una extensión del M-tree, y al igual que éste, es una estructura paginada, balanceada y dinámica que indexa datos de un espacio métrico (en particular, un espacio vectorial), resuelve consultas por rango dado un objeto de consulta, la ejecución de la búsqueda es optimizada de forma que reduce el número de datos leídos y de distancias calculadas, y es apto para datos de espacios vectoriales de alta dimensión. La extensión consiste en el tratamiento por separado de las componentes de los vectores indexados, y tiene como objetivo adaptar el algoritmo de búsqueda del M-tree a un criterio de similitud de la Recuperación de Información en la Web. Esta estructura indexa documentos Web representados en el esquema TF, emplea como distancia de indexado la norma L_2 y resuelve las búsquedas aplicando como criterio de similitud entre consulta y documento la norma L_∞.

El XM-tree logra un alto rendimiento en cuanto a calidad de resultados en un proceso de Recuperación de Información en la Web, alcanzando buenos valores de Precisión y Recall. Además, la eficiencia de las búsquedas ofrece importantes mejoras sobre los espacios vectoriales e índices invertidos. Respecto a los espacios vectoriales, el XM-tree agrupa adecuadamente los documentos permitiendo recorrer sólo los próximos a la consulta. A diferencia de los índices invertidos, revisa una fracción de la colección proporcional al conjunto de documentos relevantes. Los resultados experimentales confirmaron la concreción de ambas metas: calidad en los resultados y velocidad en la resolución de consultas.

Una extensión futura es el borrado real de datos, ya que actualmente las páginas fuera de línea siguen presentes en el índice y figuran como accesibles desde un caché, tal cual lo hace el M-tree.

REFERENCIAS

[1] R. Baeza-Yates y B. Ribeiro-Neto. (eds.). Modern Information Retrieval. *ACM Press, New York*, 1999.

[2] S. Brin y L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *In Proc. of the Seventh International World Wide Web Conference, vol. 30 of Computer Networks and ISDN Systems*, 107–117, 1998.

[3] E. Chávez, G. Navarro, R. Baeza-Yates y J. L. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321. 2001.

[4] P. Ciaccia, M. Patella y P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*, 426–435, 1997.

[5] P. Ciaccia y M. Patella. The M^2-tree: Processing Complex Multi-Feature Queries with just one Index. URL www.ercim.org/publication/ws-proceedings/DelNoe01/16_Ciaccia.pdf, 2000.

[6] D. J. Delorie. DJGPP. Copyright (c) 2003 URL http://www.delorie.com/djgpp/

[7] A. Ocsa y E. Cuadros-Vargas. DBM*-Tree: An Efficient Metric Acces Method. *In Proc. of ACM Southeast Regional Conference*, 401–406, 2007.

[8] C. Traina Jr., A. Traina, B. Seeger y C. Faloutsos. Slim-trees: High Performance Metric Trees Minimizing Overlap between Nodes. *In Proc. of 7th International Conference on Extending Database Technology*, *LNCS*, 1777, 51–68, 2000.

[9] M. R. Vieira, C. Traina Jr., F. J. T. Chino y A. J. M. Traina. DBM-Tree: Trading Height-Balancing for Performance in Metric Access Methods. *Journal of the Brazilian Computer Society*, v. 11, n. 3, p. 20 p, 2006.

[10] X. Zhou, G. Wang, J. Xu Yu y G. Yu. M^+-tree: A New Dynamical Multidimensional Index for Metric Spaces. *In Proc. of the 14th Australasian Database Conference*,161 – 168, 2003.

[11] X. Zhou, G. Wang, X. Zhou y G. Yu. BM^+-tree: A Hyperplane-based Index Method for High-Dimensional Metric Spaces. *In Proc. of 10th International Conference Database Systems for Advanced Applications*, LNCS, 3453, 398–409, 2005.